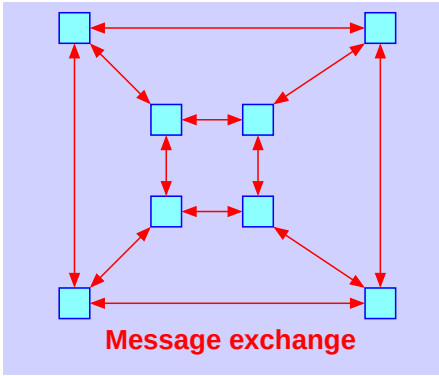
DISTRIBUTED MEMORY COMPUTING

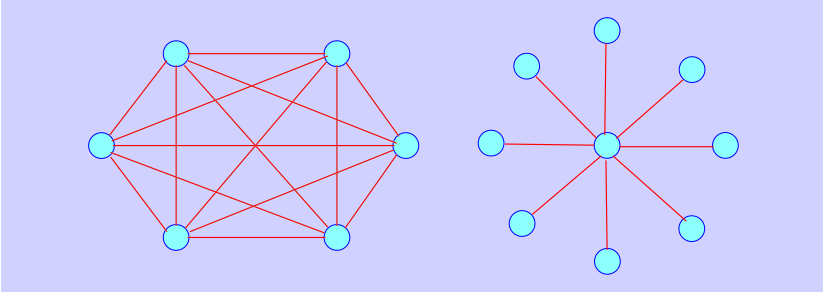
- **Interconnection networks: Static vs dynamic**
- **Simple interconnection networks**
- **Hypercubes, Fat trees, ..**
- **Routing and Embeddings**

Distributed memory systems

Main feature: No shared global memory. Processors connected in a certain way and may communicate with each other (message passing).



Completely connected and star networks

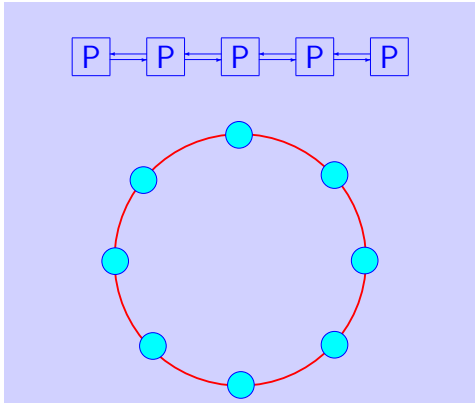


➤ Complete network : every processor connected to all others.

➤ Star network : one processor connected to all others.

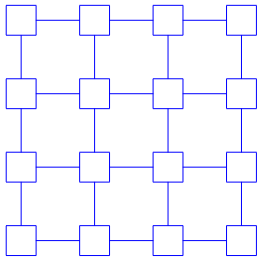
Linear arrays and Rings

➤ A Linear Array and a ring of 8 processors.
These are among the simplest networks



Two and Three-Dimensional Meshes

A 4×4 Multiprocessor Array



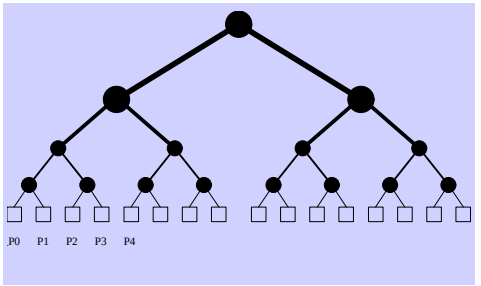
- Often implemented with wrap-around (can be viewed as a 2-D mesh of rings) – called a "Torus"
- IBM Blue-Gene uses a 3-D torus. [Also: Cray T3E from late 90s] IBM's Bluegene

Trees and Fat Trees

- Trees: nodes are linked in a tree structure – hard to program and "unsymmetric" links.
- Common remedy: double bandwidth of channels from one level to the next higher level

Fat-Tree:

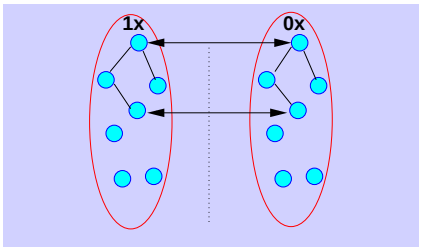
Binary tree with PEs at the leaves and communication control processors at other nodes



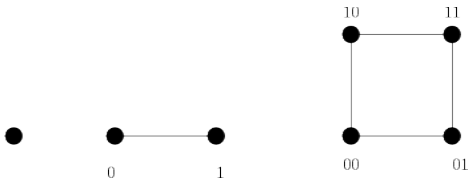
- Read about Thinking Machine's CM5 Connection Machine

Hypercubes

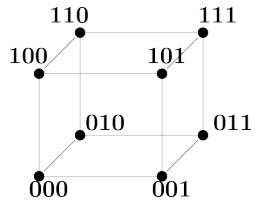
- Consists of 2^n nodes numbered with binary integers (using n bits)
- A node is connected to every other processor whose binary number differs from its own by exactly one bit.
- A hypercube of order 0 has one node.
- A hypercube of order $n+1$ is constructed by taking two hypercubes of order n and connecting their respective nodes in a one-to-one manner.



Hypercubes of dimensions 0, 1, 2, and 4

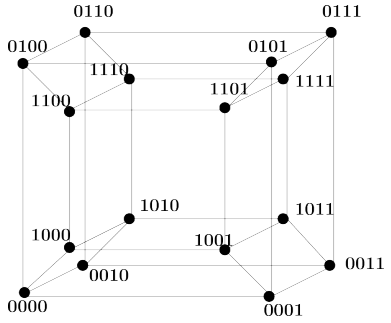


0-, 1-, and 2-cubes



3-cube

4-cube



Hypercubes (Cont.)

✍D1 How can you route a message from node A to node B in a hypercube? What is the path length?

✍D2 What is the total number of links in a hypercube?

✍D3 What is the degree (in the sense of graphs) of each node?

✍D4 What is the diameter (max. distance between any two nodes in the graph sense) of a hypercube?

Hypercubes (Cont.)

➤ The n-cube can be built recursively from lower dimensional cubes. Take two identical $(n-1)$-cubes. Label vertices from 0 to 2^{n-1}. Join every pair of vertices from the two cubes having the same label. Then relabel the nodes of the first cube as $0 \wedge a_i$ and those of the second by $1 \wedge a_i$ where a_i= label of nodes in $(n-1)$—cube and $\wedge$ denotes concatenation

➤ There are $n!2^n$ different ways in which the 2^n nodes of an n-cube can be labeled [Remember: Adjacent nodes must differ by one bit]

✍D5 Prove it [induction]

✍D6 Prove: There are no cycles of odd length in an n-cube

Performance measures for interconnection networks

Diameter: Maximum distance between any two nodes in the network (i.e., in the graph representing the network).

✍D7 What is the diameter of a 2-D grid? a 3-D grid? a 2-D torus? a 3-D torus?

Arc connectivity: minimum number of edges to remove to disconnect the network graph. Represents the "redondancy" of the network – i.e., the multiplicity of paths between nodes.

✍D8 What is the arc connectivity for rings? linear arrays? For trees? Star networks? Completely connected networks? Hypercubes?

Bisection Width minimum number of edges to be removed in order to partition the graph into two equal halves.

✍D9 What is the Bisection width for rings? linear arrays? Binary Trees? Star networks? Completely connected networks? Hypercubes?

Bisection Bandwidth: volume of communication allowed between two equal halves of network – equal to the bandwidth of the links defined in the Bisection Width

✍D10 What is the bisection bandwidth of a hypercube (assume bandwith b for each channel). What is the bisection bandwidth of a fat tree (assume bandwith b at the leaves - doubling for each higher level).

Communication costs (message passing systems)

➤ A message is sent through several nodes from origin to destination.

Three distinct costs:

1. Start-up time. Includes the time to prepare the message, handshaking, ...

2. Per-hop time. Each time a node is traversed it is handled by this node and routed (header read, message buffered, and sent out to next destination)

3. Per-word transfer time. Reflects the speed (bandwidth) of each channel. This time is the inverse of the bandwidth (in words/ sec).

Routing methods

Store-and-forward Simplest mechanism. Algorithm to send a message M of m words from A_0 to A_1 .. to A_j to A_{j+1} ... to A_l:

ALGORITHM : 1. **Store and forward send**

for $(j = 0;\ j < l;\ j++)$
  if $(j + 1 \leq l)$ *Send* M *to* A_{j+1}
  if $(j - 1 \geq 0)$ *Receive packet* M *from* A_{j-1}
End

➤ Let t_s = start-up time, t_h = per-hop time, t_w = per-word transfer time. Then total Cost:

$$T_{comm} = t_s + l * (t_h + m * t_w)$$

Packet routing Message is packetized into q small packets of size $\sim m/q$ each. Then idea is similar to pipelining.

Algorithm to send a message M split into q packets M_0, ... M_{q-1} from A_0 to A_1 .. to A_i to A_{i+1} ... to A_l:

ALGORITHM : 2. **Packet routing**

for $(j = 0;\ j < l;\ j++)$
  for $(i = 0;\ i < q;\ i++)$
    if $(j + 1 \leq l)$ *Send packet* M_i *to* A_{j+1}
    if $(j - 1 \geq 0)$ *Receive packet* M_i *from* A_{j-1}
  End
End

Simplified cost model:

➤ Neglects overhead due to packetizing, additional header info etc.

➤ First packet reaches destination in time $t_s + l\left(\frac{m}{q} * t_w + t_h\right)$

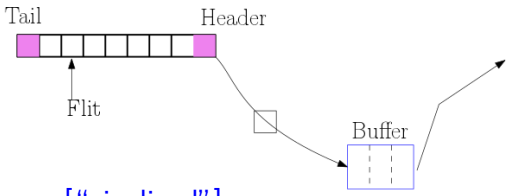
➤ $q - 1$ packets left. A new packet arrives each $\frac{m}{q} * t_w + t_h$ sec. So:

$$T_{commP} = t_s + l\left(\frac{m}{q} * t_w + t_h\right) + (q-1)\left(\frac{m}{q} * t_w + t_h\right)$$

Rewrite as:

$$T_{commP} = t_s + (q + l - 1)t_h + \left(1 + \frac{l-1}{q}\right) * m * t_w$$

Wormhole routing



➤ Packet is cut in small pieces called flow control digits or "flits".

➤ Flits transmitted one by one ["pipelined"]

➤ Header flit contains destination + routing info for all flints

➤ Buffering at flint level [not whole packet]. One buffer designated for the packet.

➤ Tail flit frees the buffer at end.

➤ Low latency, memory (buffer) efficient, ..

Cut-through routing

➤ Cut-through routing: similar but allocates buffers on a packet level instead of flit level.

Graph embeddings

Main question: We have an algorithm which uses data that is laid out according to a certain graph, for example a 2-D mesh when solving Partial Differential Equations. How can we map this data onto the network at hand if we want to optimize communication?

✍D11 How can we embed a linear array into a 2-D array? Assumptions: a $n_x \times n_y$ processor mesh – and a linear array of length $m = n_x \times n_y$.

➤ One Answer: If v_i is the i-th vertex (starting at $i = 0$) then map v_i into i_x, i_y where $i = i_x * n_x + i_y$ (integer division of i by n_x).

✍D12 Is this the best answer?

Formal definition. $G(V, E)$ mapped into $G'(V', E')$ when each vertex in V is mapped to one or more vertices in V' and if each edge is mapped to one edge or several edges (a path) of E'.

Cost measures [see text for details]

Congestion: Max number of edges mapped into an edge in E'

Dilation: Not always possible to map an edge in E exactly into an edge in E'. An edge in E is then "replaced" by a path in G'. Dilation is the maximum length of such paths.

Expansion: this is simply $|V'|/|V|$.

Gray codes and Hypercube mappings

A Hamiltonian circuit in a hypercube represents a sequence of n-bit binary numbers such that two successive numbers differ by one bit (exactly) and so that all binary numbers having n bits are represented. Binary sequences with this property are called Gray codes.

Binary reflected Gray codes:

➤ 1-Bit Gray Code: the sequence of the two one-bit numbers 0 and 1, $G_1 = \{0, 1\}$.

➤ 2-bit Gray Code: take the same sequence and insert a zero in front of each number, then take the sequence in reverse order and insert a one in front of each number:

$$G_2 = \{00, 01, 11, 10\}$$

➤ Repeat for 3 bits:

$$G_3 = \{000, 001, 011, 010, 110, 111, 101, 100\}.$$

➤ If G_i^R = the sequence obtained from G_i by reversing its order, and $0G_i$ (resp. $1G_i$) = sequence obtained from G_i by prepending a zero (resp. a one) to each element of the sequence, then

$$G_{n+1} = \{0G_n, 1G_n^R\}$$

✍13 Generate the binary 4-bit reflected Gray code and visualize the corresponding path in the picture of the 4-dimensional hupercube seen earlier in this chapter.

Gray codes and Hypercube mappings

➤ Linear arrays of length l are easy to map into n-cubes ($l \leq n$). Just use Gray codes.

➤ A ring of length l can be mapped into the n-cube when l is even and $4 \leq l \leq 2^n$.

➤ Mapping a 2-D mesh into an n-cube. Consider for example a 2-dimensional 8×4 mesh, To be mapped into into the 5-cube (64 nodes).

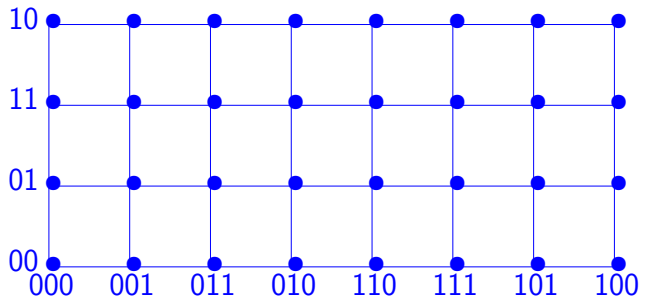
➤ Label of any node A of the 5-cube split in two parts: its first three bits and its last two bits:

$$A = b_1b_2b_3\ c_1c_2$$

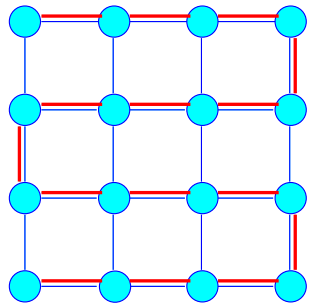
➤ Observe: When the last two bits are fixed then the resulting 2^{p_1} nodes form a p_1-cube (with $p_1 = 3$).

➤ Likewise, whenever we fix the first three bits we obtain a p_2-cube. The mapping then becomes clear.

➤ Select a 3-bit Gray code for the x-direction and a 2-bit Gray code for the y-direction. Vertex (x_i, y_j) of mesh mapped to node $b_1b_2b_3\ c_1c_2$ where $b_1b_2b_3$ is the 3-bit Gray code for x_i while c_1c_2 is the 2-bit Gray code for y_j.



Mapping linear arrays into meshes and vice-versa



➤ Consider the following mapping of a 16-node linear array into a 4×4 mesh.
➤ What are the congestion, dilation, expansion of the mapping?

✍14 Consider now the situation of an $n \times n \times n$ 3-D mesh and a linear array of n^3 nodes. How would you map the linear array into the 3-D mesh? What are the congestion, dilation, expansion in this case?

✍15 Consider the reverse situation: a 2-D mesh mapped into a linear array (reverse mapping to the one above). What are the congestion, dilation, expansion of the mapping?

✍16 Finally consider the same scenario of mapping a 3-D mesh into a linear array. What are the congestion, dilation, expansion of the corresponding mapping?